Neural Networks & Non-Destructive Test/Evaluation Methods

NEURAL NETWORKS
AND
NON-DESTRUCTIVE TEST/EVALUATION METHODS

by

JEFFREY DEAN DRAPER

A Scholarly Paper submitted to

ASSISTANT PROFESSOR IAN FLOOD

of

THE UNIVERSITY OF MARYLAND, COLLEGE PARK

for

Partial fulfillment of the Requirements for the Degree of
Master of Science in Civil Engineering
1992

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

ABSTRACT: *With today's reports of deteriorating highways and infrastructure as well as increased litigation arising from structural failures and the construction process, there is an increasing desire to employ non-destructive testing and evaluation (NDTE) methods for analyzing structural concrete members as well as other construction materials in a noninvasive manner. A major part of NDTE techniques is defect characterization, which is a typical pattern classification problem. The current state of the art for solving this problem is the application of a human expert's knowledge and experience for interpreting NDTE data. Artificial neural networks (ANNs) have shown a propensity for solving the pattern classification problem in the areas of speech and vision recognition, as well as problems in system modeling and simulation. As a result of these successful ANN applications, this paper explores the possibility of using ANNs for the NDTE defect characterization problem. Part of the solution of defect characterization entails the capability to filter what would otherwise be considered noisy data. Therefore, an ANN architecture is proposed and tested via computer simulation for the purpose of discerning between cracks and other surface defects found in photographs of defective reinforced concrete sections. Also, a basic introduction to ANNs is included along with a recommendation for continuing research.*

I. INTRODUCTION

Reinforced concrete generally performs well as long as conditions for its installation and use fall within the parameters for which it was designed. However, there have been and always will be occasions involving severe construction conditions, construction mistakes, faulty design, unforeseen disasters such as fire and flood, and/or unanticipated loads placed on structural concrete. As a result of these aforementioned inauspicious circumstances, a reinforced concrete member will show signs of distress, i.e. cracking, dusting, scaling, spauling, etc. These signs of distress will require either one or some combination of the owner, designer, and/or constructor to investigate the reinforced member to determine its strength, anticipated longevity, and need for

replacement. Indisputably, it is ideal with respect to time and money to investigate the structure without doing any damage to the member; for this reason, non-destructive test and evaluation (NDTE) methods have become popular and necessary means for analyzing the integrity of structural concrete.

Like many other scientific techniques, NDTE heavily relies on some expert to collect, graph, and interpret data. Certainly, there will never be an engineering tool which will eliminate the need for experts and good judgement. However, automation of NDTE methods would improve the speed of analyses and likely increase the frequency with which these methods are used. At a minimum, NDTE automation would allow "non-experts" who become trained on automated NDTE systems to engage in initial data collection and defect classification part of the problem.

NDTE data collection and interpretation is generally a problem in pattern classification, i.e. a true expert would almost instantly recognize that data from any given situation fits some particular problem and solution method which he has before seen. However, pattern classification lends itself poorly to traditional computing methods. Conventional computer pattern classification has involved feature extraction and clustering which more often than not requires the use of extensive prior information, such as the statistical distribution of vectors.

In the case of NDTE data pattern classification, the computerization problem is compounded with the fact that the cause for and impact of defects on materials, like many other real world functions, is extremely complex to model, requires the consideration of many factors (independent variables), and is not completely understood.

Enter the artificial neural network. An artificial neural network (ANN) is either a hardware or software system which attempts to imitate the neural structure and functioning of the biological brain (Sejnowski, Kock, and Churchland, 1988). The brain uses millions of elementary processors known as neurons which are interconnected by synapses and process sensory information (sight, sound, touch, smell, and taste), thus allowing us to perceive and ultimately react to our environment. Similar to its biological counterpart, an ANN is a massively parallel, interconnected network of simple processors which can receive and process many independent variables. The basic advantage of the ANN over other traditional serial computing techniques is the ability to take into account and process many independent variables much faster.

In addition, ANNs have shown promise for successfully performing a variety of cognitive tasks, including statistical pattern classification. Practical applications of ANNs as pattern classifiers and the need for real-time response to real-world data have led to advances in automated speech recognition, vision

recognition, robotics, and other various engineering and artificial intelligence applications.

NDTE defect classification is similar to the aforementioned real-world cognition problems for which ANNs have already been shown to have promise, i.e. problems requiring the processing of many independent variables and the classification of the result. By combining ANNs with NDTE, a significant improvement is expected in the consistency, accuracy, and ease of classifying NDTE data, i.e., a fraction of a second to classify an x-ray image or surface photograph.

II. LITERATURE REVIEW

Despite the initial skepticism in their applications and abilities (Minsky and Papert, 1969), ANNs have been shown by contemporary research as capable of solving a variety of engineering problems. The list of ANN applications includes:

- Classification of speech sounds (Lippman, 1987),
- Recognition of incoming military targets (Roth, 1990),
- Formation of text-to-phoneme rules (Sejnowski and Rosenberg, 1987),

- Deduction of the secondary structure of a protein from its amino acid sequence (Qian and Sejnowski, 1988),
- Discrimination between underwater sonar signals (Gorman and Sejnowski, 1988),
- Recognition of handwriting (Weideman, Manry, and Yau, 1989),
- Learning good moves for backgammon (Tesauro and Sejnowski, 1988),
- Performance of nonlinear signal processing (Lippman and Beckman, 1989; Tamura and Waibel, 1988),
- Prediction of the amount of energy needed to modify the thermal energy stored in a building mass (Garret, et al, 1991),
- Controlling the threshing module of a combine harvester (Garret, et al, 1991),
- Design of pump locations and rates of operation (Garret, et al, 1991),
- Recognition of machining features from a CAD drawing (Garret, et al, 1991),

With the successes of the back-propagation neural network classifier (Rumelhart, McClleland, et al, 1986) and other various ANN forms, the field of construction engineering and management has been also been targeted as an area rich with potential ANN applications (Mohan, 1990). Some proposed

applications in construction engineering and management (Moselhi, O., Hegazy, T., and Fazio, P., 1991) include:

- Predicting the bearing capacity, foundation suitability, and feasible dewatering methods based on geotechnical data,
- Estimating productivity of a crew, project performance, and cost overruns from project environment data,
- Determining project markup from various project data,
- Optimizing construction schedule and resources based on historical and current project data,
- Forecasting material costs as a function of various construction market place data.

Another application involved the use of ANNs for land-cover classification of Thematic Mapper imagery (Ritter and Hepner, 1990).

And yet another proposed application of the ANN is that of sequencing construction tasks (Flood, 1989 and 1990).

A particularly interesting concept is one that explores the combination of the pattern classification and modeling capabilities of ANNs with the heuristic rules of current expert systems; this combination of Artificial Intelligence and ANNs has interested many researchers (Gallant, 1988; Castelaz, Angus, and

Mahoney, 1987,; Derthick, 1987; Fahlman and Hinton, 1986). The marriage of the two would provide a powerful computing tool which could not only be used for scheduling construction tasks, but also could assist in identifying, classifying, and determining the probable causation and potential solutions for defects in structural concrete.

NDTE methods require an expert to interpret highly distributed, noisy information for the purpose of identifying the nature, location, and causes of defects in materials. One example of this type of application was the use of ANNs for classification of eddy current signals resulting from electromagnetic fields generated to inspect conducting materials, such as stainless steel (Udpa and Udpa, 1991). The Udpas were able to successfully use a backpropagation trained two-layer feedforward ANN to classify the eddy current signals in terms of the shape and size of the defects in their test objects. In fact, they obtained better results with the ANN than with more traditional techniques for classification of such signals.

ANNs have also been applied to the classification of signals from NDTE ultrasonic and sonic methods (Garret, et al, 1991). In this instance, a backpropagation two-layer feedforward ANN was successfully used to interpret data received from a pulse-echo hammer test to detect the presence of a flaw in a masonry wall.

Although some applications to NDTE techniques have been proposed there are many other NDTE methods which require study. In the case of structural concrete, NDTE can be used to determine structural concrete's in-place strength, uniformity of strength, cracks, delaminations, thickness, rebar location, depth of cover, and other discontinuities or conditions. Figure 1 provides an overview of NDTE methods for investigating the aforementioned as well as other conditions ("Specialized Concrete Evaluation and Testing.", 1984). A summary explanation of each of the different NDTE methods used in examining concrete is shown in Appendix I.

The first step in any NDTE investigation is a visual survey which includes observations of the surface conditions, the extent of cracking, obvious loading problems, settlement, poor drainage, chemical corrosion, etc. In structural concrete it is typically the presence of an observable physical defect which initiates any subsequent investigation. More times than not, the physical defect of concern is cracks. As can be seen by the following quote, some believe that cracks are extremely significant in the analysis of structural concrete members:

> "Cracks in structural concrete are like hieroglyphics-they are pictures that can communicate." (Gustaferro and Scott, 1990)

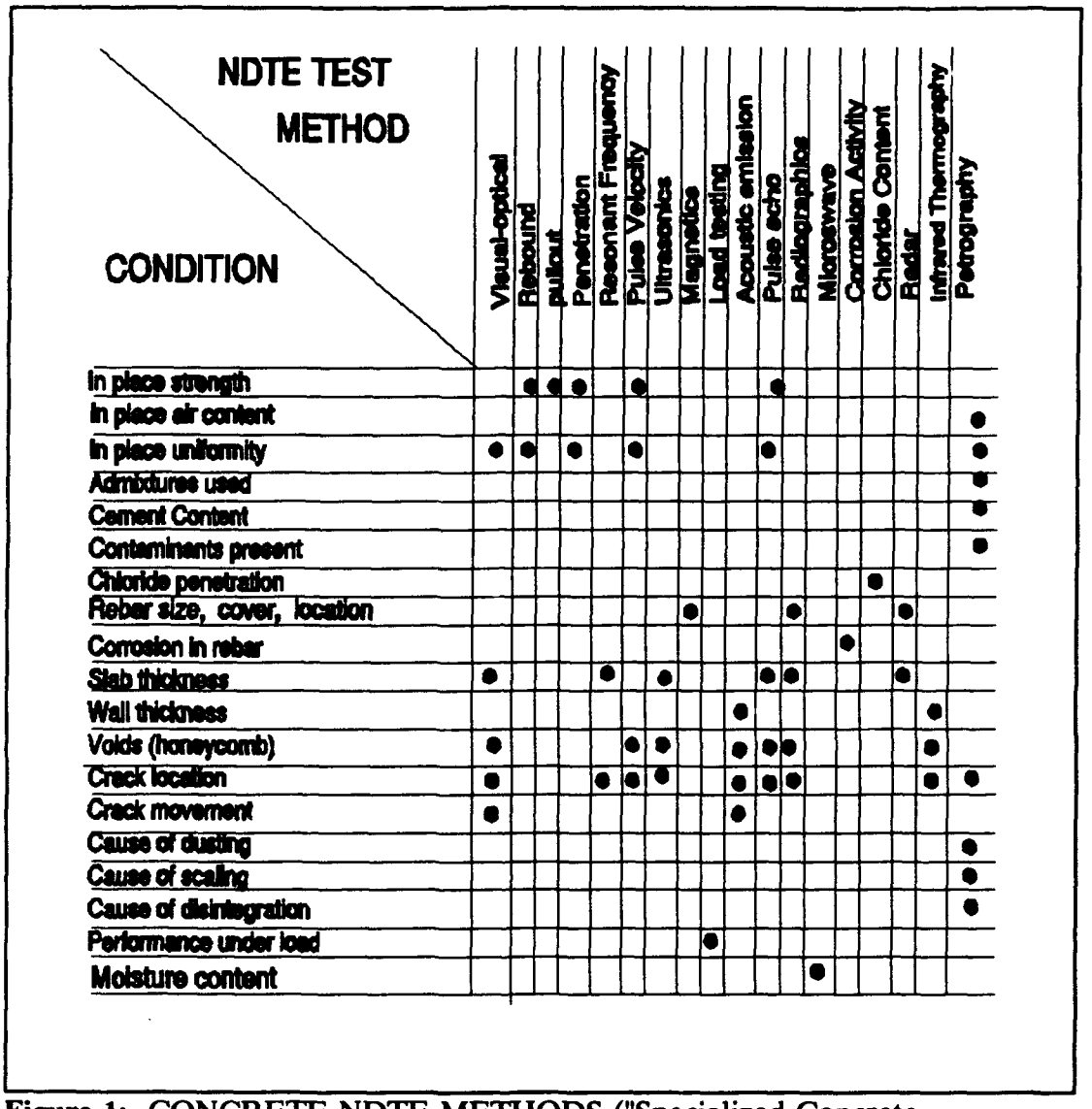


Figure 1: CONCRETE NDTE METHODS ("Specialized Concrete Evaluation and Testing", 1984)

The first observations an NDTE expert makes regarding cracks in concrete is the orientation of the cracks with respect to the orientation of stresses on the member. These observations will allow the expert to use his experience to determine why the cracking is occurring. From this assertion, it is reasonable to

conclude that any automated system used to analyze cracks in reinforced concrete must also be able to determine what type of crack is being viewed. If this is so, the design of the automated NDTE system must include image processors which are able to distinguish a crack from a hole, joint, or other surface defect. Thus, one of the first analyses of the automated system would be to filter surface defects from the image of the cracked member. Therefore, we will investigate the application of ANNs for filtering and enhancing surface photographs of cracking concrete.

III. AIMS AND OBJECTIVES OF THE RESEARCH

The primary objective of this research was to investigate the use of ANNs for the purpose of acting as a filter/enhancer for surface photographs of concrete structures. The filter/enhancer ANN would discriminate between cracks and other non-critical surface features contained in the photograph.

In order to achieve the primary objective the following interim objectives were established:

a) Review basic ANN concepts, and consider the applicability of ANNs to this specific problem, including state-of-the-art applications and research involving ANNs;

b) Analyze the problem in terms of

- Investigating the types of neural architecture and training procedures that have potential application to the problem,
- Collecting and/or developing sets of data for use in training and testing the ANN;

c) Train the ANN designed using the data developed from the previous goal;

d) Implement and evaluate the trained network on some surface photographs and evaluate the results.

IV. RESEARCH METHODOLOGY

Figure 2 shows the main stages comprising the project.

Phase A involved a literature search to provide:

- Insight into basic ANN architectures and training procedures, and
- An overview of research reflecting current state-of-the-art applications of ANNs for data modeling and pattern classification, and any specific research related to the use of ANNs for the classification of defects from NDTE data.

Phase B was concerned with developing data for use in training the ANN and design of the ANN to be used. The ANN design will be based upon the information gathered in Phase A. The data to be collected for training will be extracted from surface photographs of cracking concrete. The data was

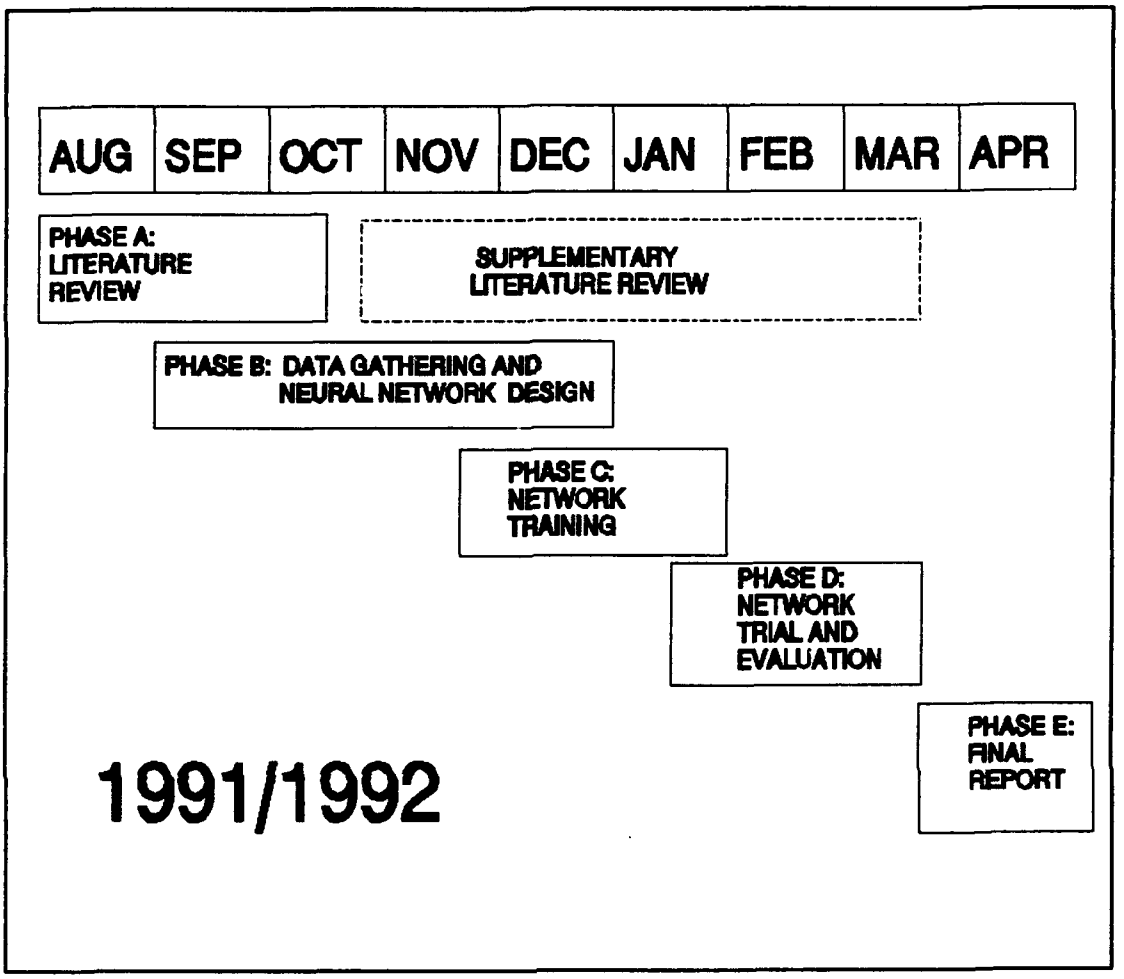


Figure 2: SCHEDULE OF WORK

imported into an IBM compatible computer with the use of a scanner. Then by use of software developed for the specific purpose, the training data was then organized and labeled.

Phase C consisted of training and completing the design of the ANN to be used as a filter for the scanned in crack images. Since ANNs are relatively new technology, little if any hardware currently exists. Therefore, the development and experimentation will be undertaken using software emulation

of the ANNs on an IBM compatible computer. Training was completed using software developed by Ian Flood, the student's faculty advisor at the University of Maryland, College Park, Department of Civil Engineering.

Phase D entailed implementing and evaluating the ANN proposed and trained in previous phases. Evaluation was based on the subjective visual comparison of the filtered/enhanced image to that of the original image. In this case, the student developed software for the implementation of the trained network.

Phase E was the culmination of the findings of this work into a final report.

V. ARTIFICIAL NEURAL NETWORK BASICS

In this section, a brief introduction is given to ANNs as relevant to this paper. A more detailed introduction to ANNs including the necessary elements of an ANN's architecture and training procedures can be found in works by other researchers (Rumelhart and McClelland, 1986; Lippman, 1989). The determination of an ANN architecture and training is highly problem dependent (Lippman, 1989; Moselhi, Hegazy, Fazio, 1990). The performance of ANNs can be significantly affected by the number of layers and the number of neurons in each layer (Rumelhart and McClelland, 1986; Huang and Lippmann, 1987, Gorman and Sejnowski, 1988). The network architecture and transfer functions

must be able to distinguish between classes of data presented to the network, be insensitive to slight variations in the input, and have a limited number of neurons to permit efficient computation and limit training data required. Excellent discussions regarding the types of ANN classifiers, their memory requirements, and performance characteristics have been written (Rumelhart, McClleland, et al, 1986; Lippman, 1989; Bailey and Thompson, 1990).

Architecture. Figure 3 is a simple three-layer network consisting of an input layer, a hidden layer, and an output layer. The ANN is made up of neurons (the nodes), and connections (the lines connecting the nodes). The architectures (physical configurations) of neural networks are described by the number of layers of neurons in the network, the number of neurons in each layer, and their connections with one another. There are two general classes of ANN architectures:

- feedforward, or nonrecurrent, and
- recurrent.

The ANN shown in Figure 3 is a feed-forward network which means the flow of information is in only one direction. Recurrent networks differ from feed-forward networks in that they contain feedback connections between layers or between neurons in the same layer.

Every neuron performs both a summation function (Σ) and a transfer function (f). Each neuron receives a set of inputs (real or discrete data) from

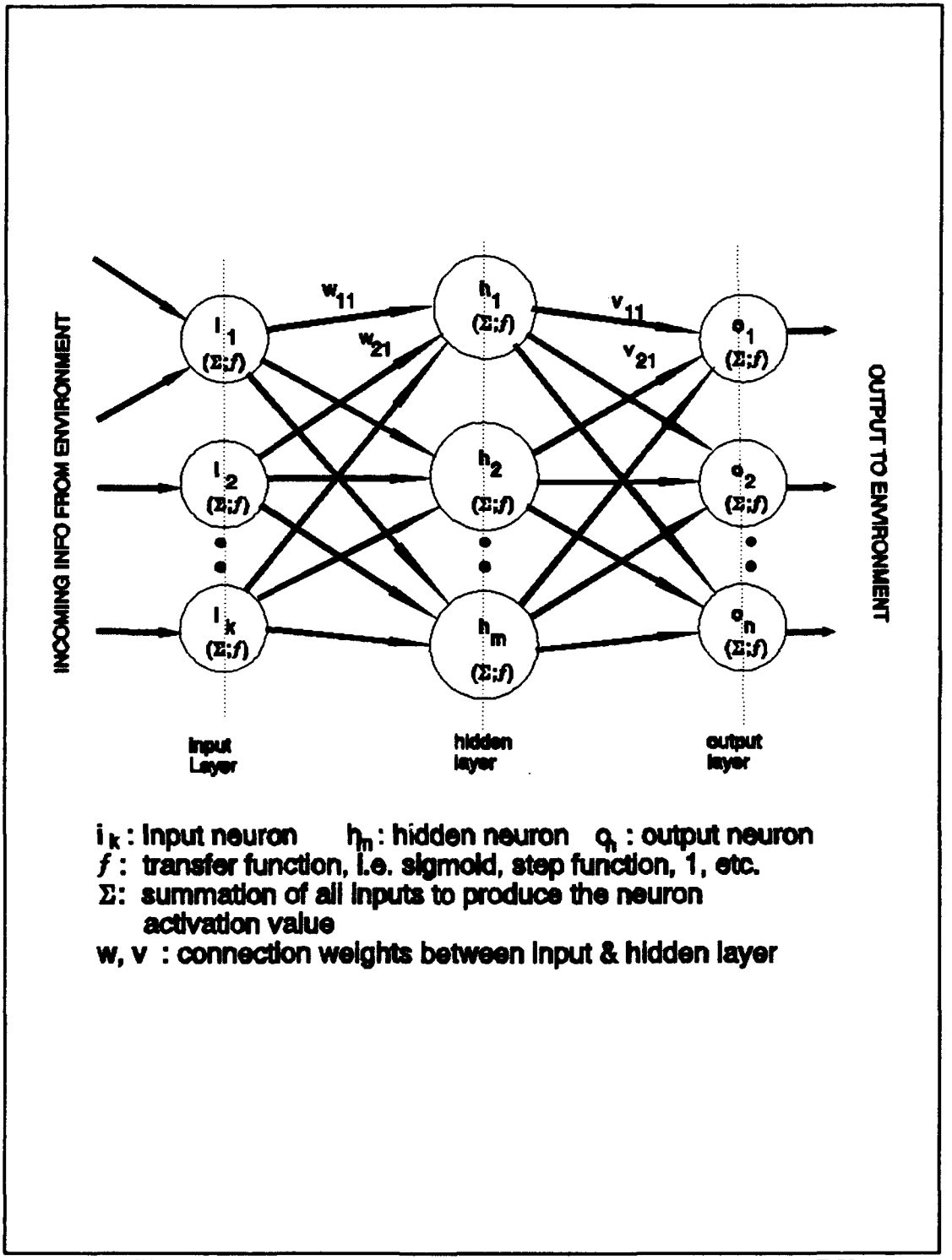


Figure 3: THREE-LAYER FEEDFORWARD NEURAL NETWORK

the preceding layer of neurons, with the exception of the input layer which receives inputs from the network's interface with its environment. A neuron adds its incoming inputs together to produce a sum known as that neuron's activation value, ***a***. The activation value, ***a***, is then operated upon by the neuron's transfer function, ***f***, to produce the neuron's output. The transfer function can be just about any function including the identity function. Hidden layer neurons commonly have a transfer function of one of the forms shown in Figure 4. However, the transfer function can be just about any mathematical form which provides an output between 0 and 1 and satisfies both the needs of the network and the problem being solved. Sometimes, the transfer function passes information out of the neuron only if **a** is greater than some value known as the threshold, **t**; otherwise the output of the neuron is set equal to zero. If the transfer function is a discrete threshold function, the output will be 1 if $\mathbf{a} \geq \mathbf{t}$ and 0 if $\mathbf{a} < \mathbf{t}$. If the transfer function is continuous, the output of the neuron is a real number between 0 and 1. Transfer functions can also contain other constants such as bias, **b**. Biases and the other constants position the resulting transfer function relative to the activation value axis.

The output of each neuron is operated upon by the weights, **w** or **v**, on the connections to the next layer of neurons. The connection mathematics usually takes the form of a simple multiplication of the neuron's signal by the connection weight, however other forms are possible. This operated upon signal

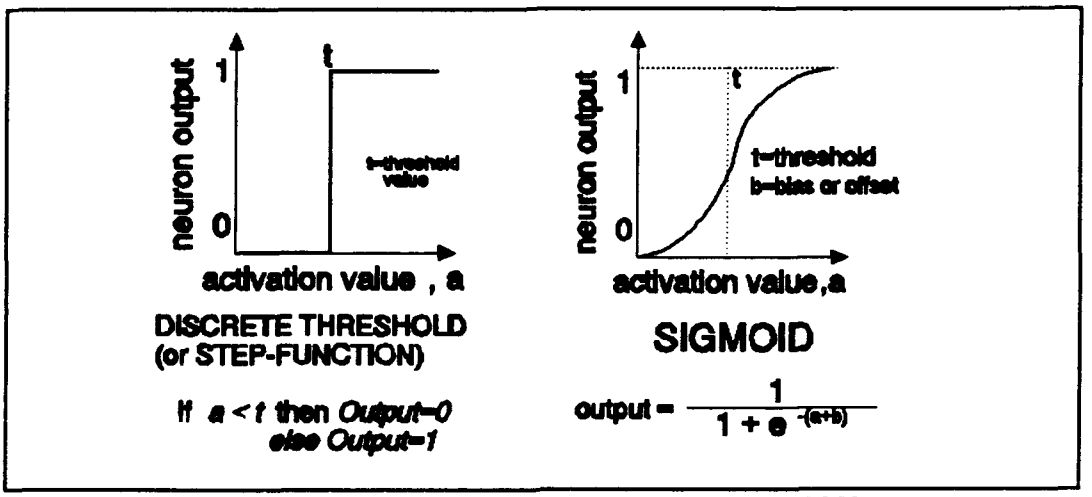


Figure 4: COMMON NEURON TRANSFER FUNCTIONS

becomes the input to the next layer of neurons, or the output from the ANN to the environment.

Training. Training or learning is the process of adapting the connection weights, thresholds, and other variables of the ANN in response to training data being presented to the network. The **training method (or learning rule)** is the mathematical relationship that generates an ANN's desired output for a particular set of inputs while setting the coefficients (**b, t, w, v,** etc.) in the neuron's local memory.

Training can take one of three basic forms as shown in Figure 5 (Lippman, 1989). The three types of training methods are "supervised", "unsupervised", and a combination of the supervised and unsupervised.

Supervised training implies that the ANN is presented input data (called patterns) along with the desired output (called targets), and simply organizes its

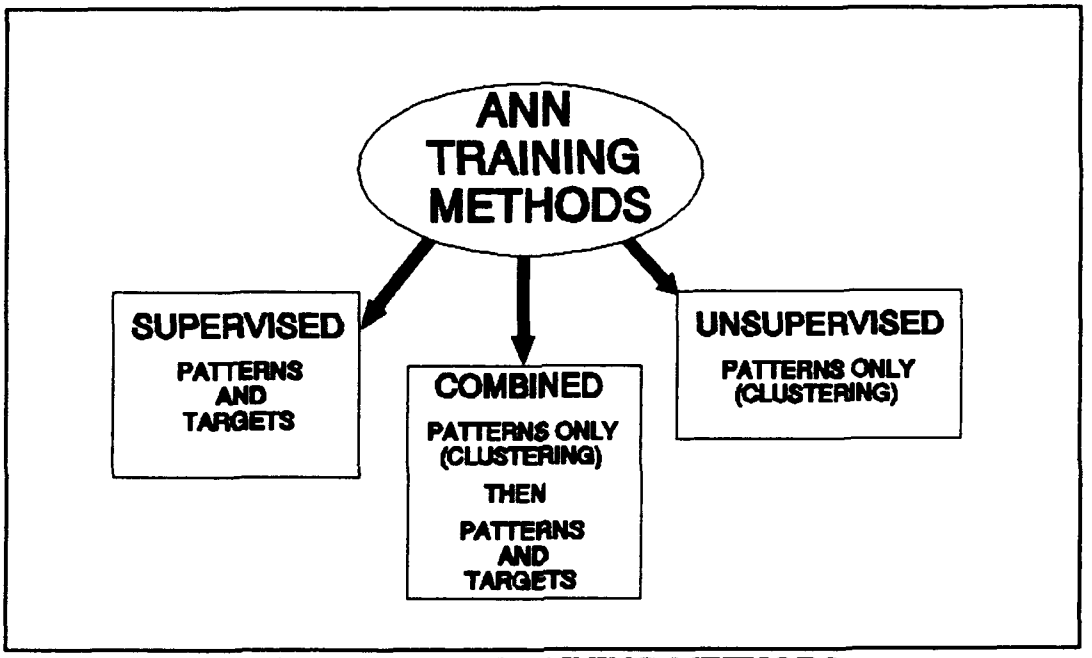


Figure 5: NEURAL NETWORK TRAINING METHODS

internal connection and transfer function parameters in order to make the actual ANN outputs meet the expected targets. In backpropagation training for example, the network cycles patterns of inputs attempting to achieve connection weights and neuron responses that modify the pattern of inputs to those of the desired targets. At the end of each cycle, the difference in the output signal and the desired target acts as feedback to the network to modify the connection weights and transfer function thresholds for the next cycle.

Unsupervised learning allows the network to develop what are referred to as internal clusters from the patterns it is presented (no targets are presented), i.e., the network simply places its results into groupings of outputs.

The combination method of training allows the ANN to first organize the internal clusters as with the unsupervised training technique, to be followed by labeling the clusters with targets and sequentially retraining the network. An advantage of the combined training method is that it can simplify data collection and reduce expensive and time consuming data labeling.

VI. APPLICATION DEVELOPMENT OF THE ANN FOR VISUAL IMAGE ENHANCER AND FILTER PROBLEM

The ANN application development procedure consisted of three phases as shown in Figure 6: design, training (learning), and implementation and observation.

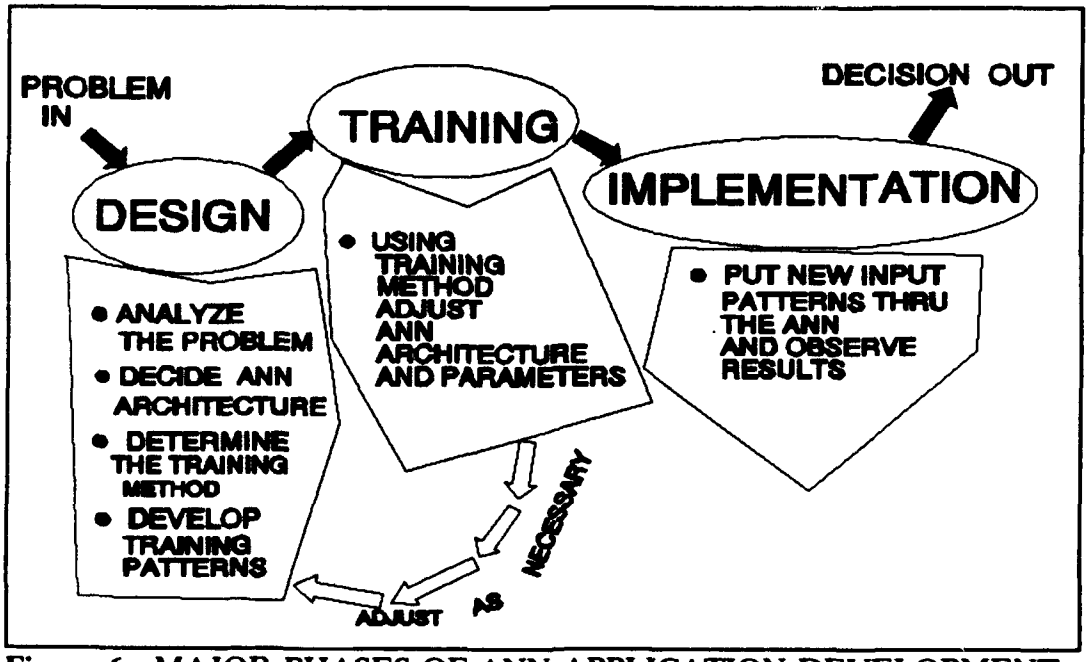


Figure 6: MAJOR PHASES OF ANN APPLICATION DEVELOPMENT

Design Phase. In the design phase, we first needed to analyze the attributes and parameters of the problem.

Humans classify imagery by using both spectral and spatial associations ingrained in biological neural networks which connects their eyes to the area of their brain controlling vision. However for this problem, consideration of spatial associations was sufficient since our computerized image was a two dimensional image having no spectral information and consisting of an image of pixels which were either "on" or "off".

The problem was then reduced to distinguishing between the two-dimensional image of a crack and another type of surface defect. As far as our computerized image was concerned, the difference between a crack and a surface defect was that a crack will form a continuous line of pixels that are "on" while a surface defect will be a few "on" pixels surrounded by "off" pixels.

By the use of an image sampling device consisting of a pixel window (an array of pixels, i.e. 3 x 3, 5 x 5, 7 x 7, 9 x 9, etc.) which would methodically scan the image, the network theoretically would be able to assimilate data of spatially adjacent pixels in both the training and implementation phases of this application. In effect we considered each element of the pixel window to be an input to the network with the resulting output to be a logical response, i.e. "true" if the pixel window represents a segment of a crack or "false" otherwise.

For this visual filter/enhancer problem, the basic ANN paradigm chosen was that of a back-propagation trained feed-forward network using neuronal discrete threshold activation functions. The criteria for the network chosen were the following:

- High network accuracy
- High interpolative performance
- Since boolean input and output are desired, the transfer function should be one that provides either a 0 or a 1 depending on the comparison of the neuron's activation value to that of some threshold.
- No incremental learning or real time performance desired
- Training time was not an issue since relatively few examples were expected to be used.
- Relatively low memory requirements since the amount of data generated by graphic images is considerable and could easily overburden or crash even today's powerful desktops computers.

Hidden layers of neurons were considered necessary because of the original skepticism from experiments with the neural network form known as the perceptron (Minsky and Papert, 1969). Although others researching the ANN application for NDTE data classification have used networks with two-hidden layers (Udpa and Udpa, 1991; Garret, et al, 1991), none have given any specific reasons for their choice of architecture. Some constructive proofs have

demonstrated that two hidden layers are sufficient to form arbitrary decision regions using multilayer ANNs with discrete threshold functions, while others have more recently shown that multi-layer ANNs with only one hidden layer (and no specific number of hidden neurons) could form complex disjoint and convex decision regions (Lippman, 1991). Since our particular problem only deals with a relatively simple decision region, having a value of 0 (false) or 1 (true), only one layer of hidden neurons was considered adequate. Additionally, by having only one hidden layer of neurons, the training method of backpropagation would be able to be used to determine the correct number of neurons to reduce the error between actual and target outputs to zero. The number of hidden neurons was found by setting the parameters for the first neuron and cycling through the training patterns and targets until a best fit could be found. Then a second neuron was added to fit the errors remaining from the first cycle, and so on until the error is reduced to zero or some acceptable value.

The method of scanning the image with the pixel window also became a factor in the application of the ANN. The scanning technique had to operate in such a way that the final enhanced/filtered image would be the same size as the original image, and at the same time avoid the loss of information. Based on the way the retina receives visual information, we determined each pixel window sample should overlap its adjacent pixel window sample. Therefore, we scanned the original image by starting our pixel window in the first row leftmost element

of the image and then moved the pixel window one pixel (element) to the right until the end of the image row was reached; at that point the pixel window was moved down to the next row and the process repeated until the last row rightmost element was reached.

Our next effort in the design phase of the problem of filtering voids from our image involved the development of training patterns and their targets.

We decided upon using 3 x 3 and 9 x 9 pixel window samplers for the experiment. The 3 x 3 sampling window represents the smallest window which would provide any spatially associative information. The 9 x 9 sampling window represents a sizeable increase over the 3 x 3 in respect to the possible number of combinations of pixels which could represent a crack configuration. A square sampling window having an odd number of elements on each side was considered necessary because we defined a crack image as a line of "on" pixels passing through the centroid of the sampling window.

In the case of the 3 x 3 pixel sampling window, a vector with 9 elements, was created. Therefore the number of permutations of "on" and "off" pixels was $2^9=512$. These 512 combinations were generated using a Pascal program PATTERNS3 as shown in Appendix II. The philosophy on assigning the target values was to label the target "true" for any combination of the sampling window which had the center pixel of the sampling window "true" and had at least two other pixels of the sampling window "on" but not adjacent to each other.

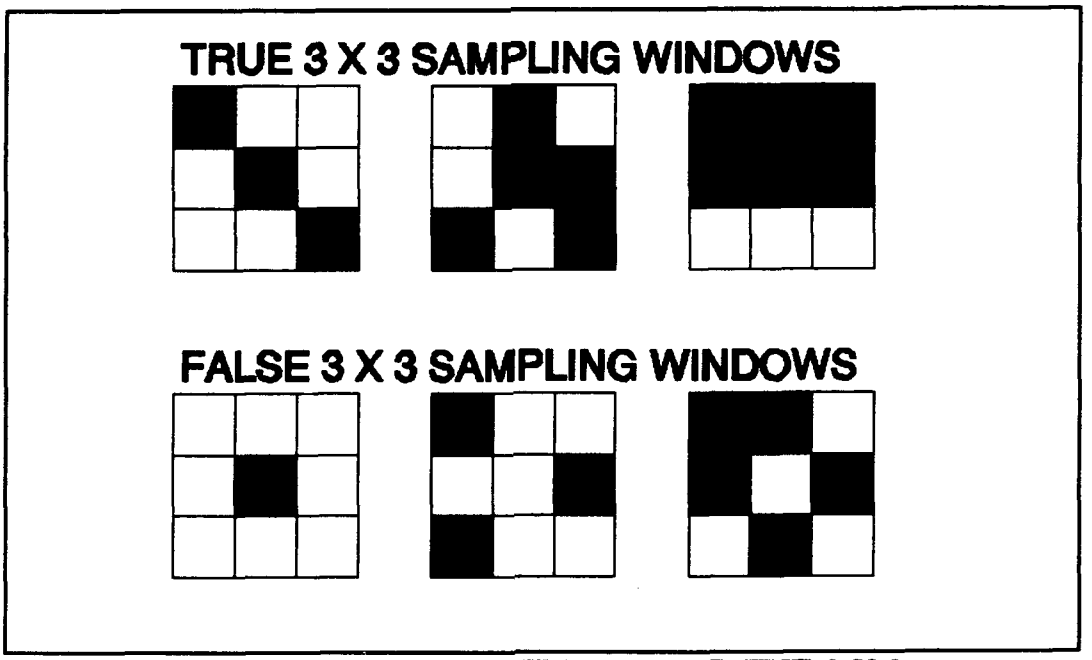


Figure 7: SAMPLE TRAINING PATTERNS FOR THE 3 X 3 SAMPLING WINDOW

Otherwise, the target was set to "false". A few examples of the true and false sampling windows are shown in Figure 7.

In the case of the 9 x 9 sampling window, a vector with 81 elements, was created. Therefore the number of permutations of "on" and "off" pixels is $2^{81} = 2.4179 \times 10^{24}$. Although this is a finite number of combinations, only fewer than 800 were to be used because of computer memory and speed limitations, and the ability of the ANNs to generalize. We used about 600 patterns. The 600 input patterns and their respective target values were generated using the Pascal Program PATTERNS (APPENDIX III). In essence, PATTERNS randomly selects training patterns from a scanned in photograph of cracking

concrete and generates a visual image of the pattern on the display. Then the user enters the value of the target based on his subjective determination that the pattern shown represents a crack segment. A few examples of true and false sampling windows are shown in Figure 8.

Training (Learning) Phase. Having the patterns and the target values established, the Pascal Program BINARYHAM (developed by I. Flood) was used to train the network and determine the number of neurons in the one hidden layer. BINARYHAM uses backpropagation as the training method and

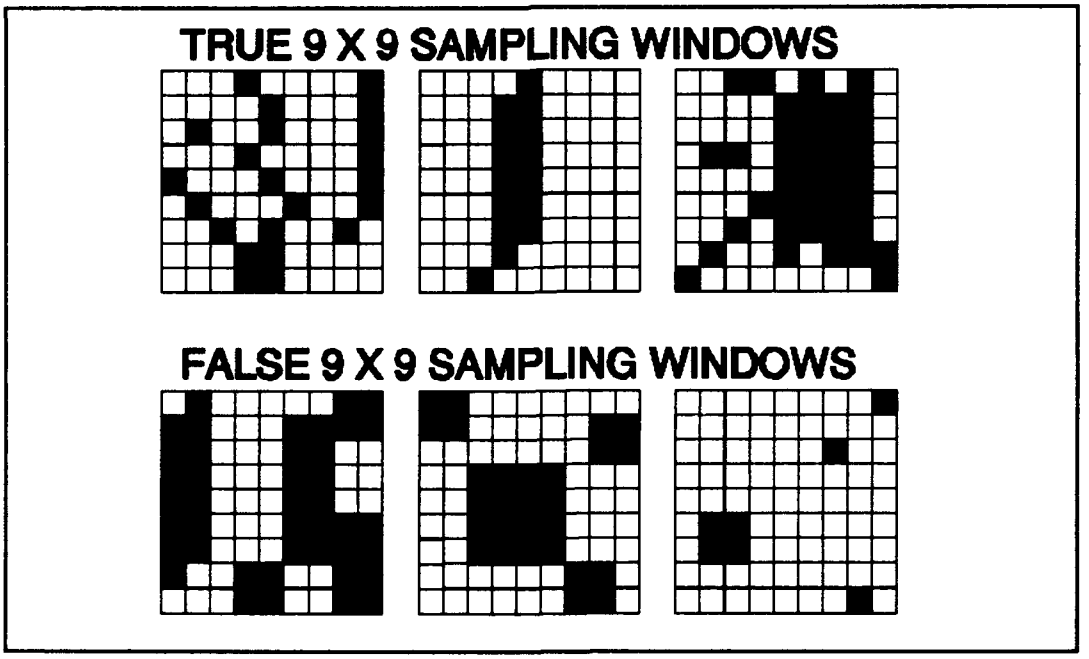


Figure 8: EXAMPLE TRAINING PATTERNS FOR THE 9 X 9 SAMPLING WINDOW

determines the number of neurons required to make the error between actual and desired outputs equal to zero. The results of the program BINARYHAM are a set of templates (one for each neuron in the network of the same length as the vector developed by the original sampling window) and a set of threshold values, one value for each neuron in the network. The results of the program BINARYHAM for the 3 x 3 sampling window are shown in APPENDIX IV and those for the 9 x 9 sampling window are shown in APPENDIX V. The neuron template elements are boolean values (0=false, 1=true). The thresholds are integer values which represent the required activation value of the incoming inputs to a neuron in order to make the network "fire". The basis for the neuron operation is that an incoming pattern from the image being filtered would be compared to a template pattern and the differences summed; if the sum of the differences exceeds the threshold, then the neuron would fire thereby producing an output of 1 (true), otherwise the output would be zero. A neuron output of 1 (true) indicates the incoming pattern represents a segment of a crack. In summary, the 512 training patterns for the 3 x 3 sampling window resulted in 52 hidden neurons, and the 600 training patterns for the 9 x 9 sampling window resulted in 38 neurons. Since the ANN has the ability to generalize, this disparity between the number of training patterns and the number of hidden neurons was in line with our expectations. The training time for both sets of input patterns was seconds using an IBM compatible 386-33MHz.

Implementation and Evaluation Phase. In the final phase of ANN application development, the templates and the thresholds developed by the Program BINARYHAM were recalled to form the ANN to be used. The program FILTER was used to scan the incoming image with the sampling window and to output a filtered/enhanced image. The measure of effectiveness of the filter/enhancer was a visual comparison between the originally scanned image and the ANN generated image.

Examples of the network output using the 3 x 3 network filter are shown in Figure 9, and those using the 9 x 9 network filter are shown in Figure 10.

In the case of the 3 x 3 network filter, we were able observe some meager filtering capabilities. Improvement was gained when the image was iteratively ran through the program FILTER several times. Another observation was that noticeable information was lost at the edge of the image and at the ends of crack segments.

In the case of the 9 x 9 network filter, the filtering/enhancing capabilities were much greater. The 9 x 9 filter was able to eliminate large quantities of defects from the incoming image, however, it also deleted significant amounts of crack information, especially at points where cracks intersected one another.

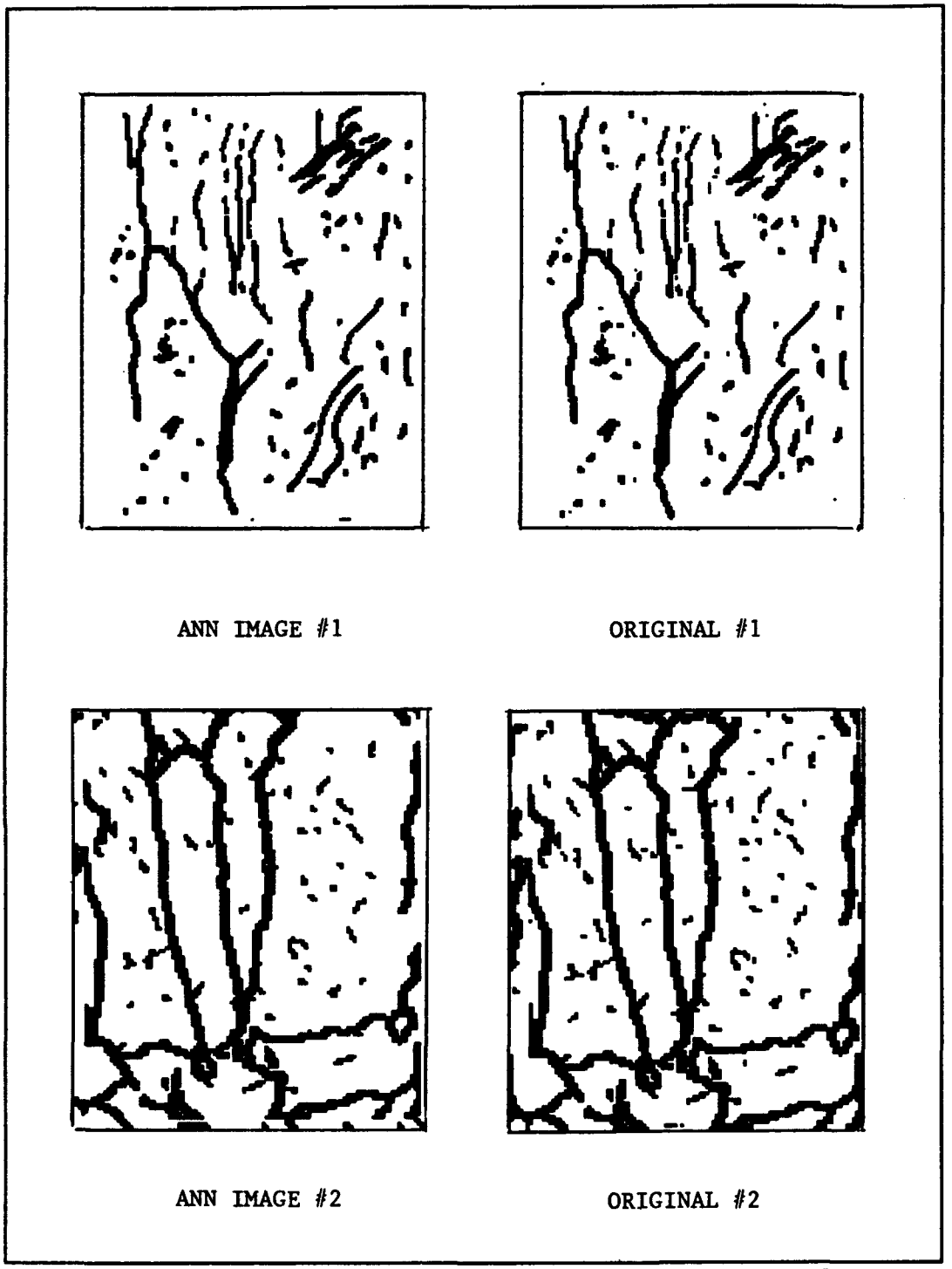


Figure 9: ORIGINAL VS. ANN FILTERED/ENHANCED IMAGES FOR THE 3 X 3 SAMPLING WINDOW

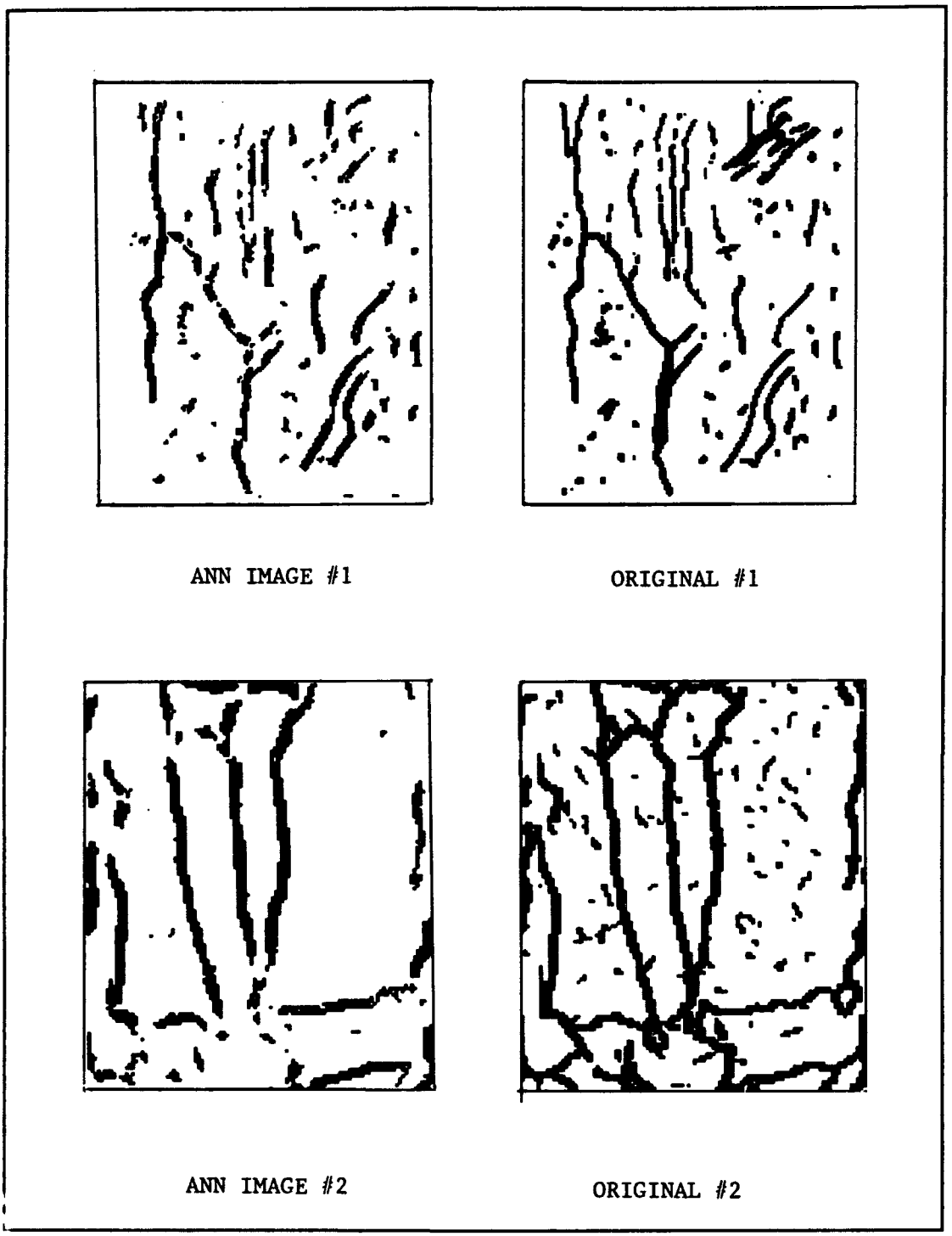


Figure 10: ORIGINAL VS. ANN FILTERED/ENHANDED IMAGE FOR 9 X 9 SAMPLING WINDOW

VII. CONCLUSIONS & RECOMMENDATIONS

Conclusions from examples shown. All in all, both ANN filter/enhancers demonstrated the ability to successfully eliminate unwanted information from the original photographic images. In the case of the 3 x 3 filter/enhancer, the ANN demonstrated its ability to filter small pieces of noise from the image. However, the 3 x 3 ANN filter was limited in the size of the surface defect that it would effectively filter from the incoming image. Additionally, because of the way in which the training patterns were labeled, the 3 x 3 filter/enhancer slowly ate away at the tips of cracks as well as other surface defects.

The 9 x 9 ANN filter/enhancer demonstrated greater abilities to extract large amounts of defect information from the image, but at the expense of losing pieces of the cracks.

In both cases, the deficiencies with the ANN filter/enhancer operation were directly traceable to the training patterns and the image scanning methodology specifically used for this experiment.

Yet, even with considering the deficiencies of the two ANN filter/enhancers used, this experiment validates the ability of this ANN approach for improving concrete crack images for the ultimate purpose of classifying the cracks observed.

Recommendations for future work for this application. A more thorough investigation is currently being undertaken to find a combination of different filters and scanning techniques which will be able to overcome the problems left remaining with this experiment.

One example, would be to start with a 9 x 9 filter to eliminate large surface defects shown in the image, then follow with another filter which would subsequently complete the missing crack information. Then apply a 3 x 3 filter to clean-up the image.

Another proposed solution is to use a scanning technique involving the use of a large sampling window where the network weights information at the center of the window more heavily than at the edge.

By experimenting with different sampling window sizes, rules for establishing the training patterns, and various scanning techniques, this researcher is confident that an acceptable filter for enhancing the photographic images of concrete cracks can be found.

Recommendations for future applications of ANNs to NDTE data modeling and classification problems.

In the long run, we want to be able to combine the photograph filter/enhancer ANN with another ANN classifier which would then be used to classify the type of crack being studied.

A follow on project could be the study of ANN applications for other NDTE techniques (say the classification or modeling of acoustic signature recognition or radiography). On a grander scale, another possibility is to combine the crack classifier ANN with one of the other NDTE methods such as acoustic signature recognition to create and analyze a 3-D image of the cracked structural member.

The pinnacle of research along these lines would be the development of hardware neural networks which could be programmed and placed in the field for a technician to use in quality control or investigation of structural concrete.

VIII. ACKNOWLEDGEMENTS

I am forever in debt to Ian Flood without whose guidance and insight this paper would not have been possible.

APPENDIX I. SPECIFIC TESTS USED TO EVALUATE CONCRETE(Extracted from "Specialized Concrete Evaluation and Testing", 1984)

Standard methods for conducting some of the tests briefly described here are available from the American Society for Testing and Materials (ASTM) and the American Concrete Institute (ACI). These standards are noted in the descriptions

Visual-optical: Includes visual inspection for cracks, roughness, color variations, corrosion, deterioration, and similar defects that can be detected with or without the use of optical aids such as low-power magnifiers. Also includes measurement of differential structural movements and use of fiber optics to detect internal cracks, voids or flaws.

Rebound: Use of a spring-driven steel hammer to determine the uniformity of in-place concrete strength and to delineate zones or areas of poor quality or deteriorated concrete in structures. Standard test method is described in ASTM C 805-79.

Pullout: Determination of the pullout strength of hardened concrete by measuring the force required to pull an embedded metal insert and the surrounding section of concrete from a concrete mass. Pullout strength is related to other strength test results. Standard test method described in ASTM C 900-82 is for situations where the metal insert is embedded in fresh concrete. A modification of this method permits its use even if inserts have not been cast in place.

Resonant frequency: Measurement of the fundamental frequency of concrete for use in determining uniformity. (ASTM C 215-60 but this method is used primarily in laboratory tests).

Pulse velocity: Measurement of the time of travel of a pulse or train of waves through concrete to determine the uniformity, to indicate changes in characteristics or to survey structures to estimate the severity and the extent of deterioration, cracking or both. (ASTM C 597-71).

High-energy ultrasonics: Measurement of time it takes a high-energy pulse to travel through concrete. Used primarily to measure thickness.

Magnetic: Use of a portable magnetic electric test device, either a cover meter to measure the depth of concrete cover over rebars or a pachometer to measure both the cover and size of reinforcing bars.
Load testing: Application of a test load to a structure in a manner that stimulates the load pattern under design conditions. Test failure indications such as Acoustic Emission: Acoustic emission techniques detect, process, and record "noise" emitted by a structure that is placed under load. They are used primarily to detect crack growth but can also be used for determining the location of a crack.

Pulse Echo: Measures velocity of a reflected pulse generated by mechanical impact. Detects and delineates internal discontinuities in concrete and, with interpretation, identifies the nature and orientation of the discontinuities.

Radiographics: Use of X-rays or gamma rays, primarily to determine the size and location of reinforcing bars. Can also be used to detect voids or other flaws.

Microwave absorption: Measurement of microwave absorption by the concrete to estimate the moisture content or the quality of the concrete.

Corrosive activity: Detection of active corrosion, by methods such as direct measurement of current flow (half-cell potential) as described in ASTM C876-80.

Chloride content: Chloride ion monitoring by testing crushed samples of concrete from cores obtained in the field.

Radar: Use of electromagnetic impulse signals to detect voids beneath pavements or slabs on ground or to measure slab thickness. Can also be used to measure size and location of reinforcement.

Infrared Thermography: Use of selective infrared frequencies to identify heat patterns characteristic of certain defects. One use is to detect delaminations in bridge decks or pavements.

Floor Flatness: Measurement of floor flatness using a profile graphing instrument or an instrument that gives digital readouts of elevation differences.

Petrography: Use of microscopic examination, sometimes in combination with other techniques, to examine samples of concrete. Features that can be evaluated include denseness of cement paste, depth of carbonation, occurrence of bleeding, presences of leakage, excessive deflection or even structural failure are monitored and measured visually or with detection devices. Procedures for load testing are described in ACI 318-83, Part 6. contaminating substances, air content, and other properties. Standard recommended practice is give in ASTM C 856-77.

APPENDIX II. PROGRAM PATTERN3 FOR THE DEVELOPMENT OF TRAINING PATTERNS FOR A 3 X 3 SAMPLING WINDOW

```
program pattern_generator;

uses dos,crt,graph;

const
  max_patterns=800;
  max_inputs=81;
  max_size4_rows=200;
  max_size4_cols=200;
  max_size1_rows=9;
  max_size1_cols=9;

type

  boolean_array_size1=array[1..max_size1_rows,1..max_size1_cols] of boolean;
  boolean_array_size3=array[1..max_patterns,1..max_inputs] of boolean;
  boolean_array_size4=array[1..max_size4_rows,1..max_size4_cols] of boolean;
  boolean_col_vector=array[1..max_inputs] of boolean;

var
  criteria:                byte;
  data_count:               integer;
  i,j,k,l,m,n,x,y:         integer;
  max_pattern_count:        integer;
  pattern:                 ^boolean_array_size3;
  pattern_count:            integer;
  pattern_file:            text;
  repeat_check:             byte;
  repeat_count:             byte;
  samp:                    boolean_col_vector;
  samp_rows,samp_cols:      byte;
  samples_remaining:        integer;
  subj_file_name:          string;
  subj_rows,subj_cols:      byte;
  subj:                    ^boolean_array_size4;
  subj_file:               file of boolean;
  sum:                     byte;
  target:                  byte;
  target_file:             text;


{********************** procedure beep ********************************}
procedure beep;
 var
   i:    byte;
   Freq: integer;
   Time: integer;
 Begin
   Freq:=250;
   For i:=1 to 3 do begin
     Sound(Freq);
     Time:=250;
     Delay(Time);
     Nosound;
```

```
      Time:=60;
      Delay(Time)
    End;
  End;

{********************** procedure write output ************************}
 procedure write_output;
  var
    i,j,k: byte;
  begin
    k:=0;
    for i:=1 to samp_rows do begin
      for j:=1 to samp_cols do begin
        k:=k+1;
        if pattern^[pattern_count,k]=true
         then write(pattern_file,' ',1)
         else write(pattern_file,' ',0);
      end;
    end;
    writeln(pattern_file);
    writeln(target_file,target);
  end;


{************************ procedure summarizer ************************}
 procedure summarizer(var sum: byte);
  var
    i,j,k: byte;
  begin
    sum:=0;
    k:=0;
    for i:=1 to samp_rows do begin
      for j:=1 to samp_cols do begin
        k:=k+1;
        if pattern^[pattern_count,k]=true then sum:=sum+1;
      end;
    end;
  end;


{********************** procedure sample taker ************************}
 procedure sample_taker;
  var
    i,j,k,l,n: integer;
  begin
    randomize;
    i:=1+random(subj_rows-samp_rows);
    randomize;
    j:=1+random(subj_cols-samp_cols);
    n:=0;
    for k:=i to i+samp_rows-1 do begin
      for l:= j to j+samp_cols-1 do begin
        n:=n+1;
        samp[n]:= subj^[k,l];
      end;
    end;
  end;
```

```
{********************** procedure pattern_setter ************************}
procedure pattern_setter;
 var
   i,j: byte;
   k:   integer;
 begin
   pattern_count:=pattern_count+1;
   k:=0;
   for i:=1 to samp_rows do begin
     for j:=1 to samp_cols do begin
       k:=k+1;
       pattern^[pattern_count,k]:=samp[k];
       if pattern^[pattern_count,k]=true then write(chr(176),chr(176))
                                 else write(chr(178),chr(178));
     end;
     writeln;
   end;
   writeln('Exiting pattern_setter....pattern_count:=',pattern_count);
 end;


{********************** procedure repeat checker ************************}
procedure repeat_checker(var repeat_check:byte);
 var
   i:          integer;
   j,counter: byte;
   out_count: integer;
 begin
   if pattern_count=1
    then out_count:=pattern_count+1
    else out_count:=pattern_count;
   i:=0;
   repeat
     i:=i+1;
     counter:=0;
     for j:=1 to samp_rows*samp_cols do begin
       if pattern^[i,j]=samp[j] then counter:=counter+1;
     end;
    until (counter=samp_rows*samp_cols) or (i=out_count);
    if counter=samp_rows*samp_cols
     then repeat_check:=0
     else repeat_check:=1;
 end;


{***************************** target decider ***************************}
procedure target_decider;
 var
   i,j: byte;
   k: integer;
 begin
   write('Time for a target decision....Enter the Target [ 0 or 1]:');
   {beep;}
   readln(target);
 end;
```

```
{********************** target_setter procedure **********************}
 procedure target_setter;
  var
    midpoint: byte;
  begin
    sum:=0;
{    midpoint:=trunc(samp_rows*samp_cols/2+0.5);
    if (pattern^[pattern_count,midpoint]=false)
     then target:=0 else summarizer(sum);
    if (pattern^[pattern_count,midpoint]=true) and (sum<criteria) then target:=0;
    if (pattern^[pattern_count,midpoint]=true) and (sum>=criteria)
     then target_decider;    }
    target_decider;
    writeln('target setter complete.....target=',target);
    readln;
  end;


{************************ procedure rotater ****************************}
 procedure rotater(var pattern_count:integer;samples_remaining:integer);
  var
     i,j,rotation_count: byte;
     k: integer;
     matrix,rotation: boolean_array_size1;
     max_rotations: byte;

  begin

{....Set loop for max_rotations}
    if samples_remaining<3
     then max_rotations:=samples_remaining
     else max_rotations:=3;
    for rotation_count:=1 to max_rotations do begin

{....Set pattern to square matrix}
     k:=0;
     for i:= 1 to samp_rows do begin
       for j:=1 to samp_cols do begin
          k:=k+1;
          matrix[i,j]:=pattern^[pattern_count,k]
       end;
     end;

{....Take the rotation of the square matrix}
     for i:=1 to samp_rows do begin
       k:=samp_rows;
       for j:=1 to samp_cols do begin
          rotation[i,j]:=matrix[k,i];
          k:=k-1;
       end;
     end;

{....Convert rotation to a new pattern}
     writeln;
     pattern_count:=pattern_count+1;
     writeln('In rotater,  Pattern number: ',pattern_count);
     k:=0;
     for i:=1 to samp_rows do begin
```

```
      for j:=1 to samp_cols do begin
         k:=k+1;
         pattern^[pattern_count,k]:=rotation[i,j];
         if pattern^[pattern_count,k]=true then write(chr(176),chr(176))
                                  else write(chr(178),chr(178));
      end;
      writeln;
    end;

{....set the value of Target}
    target:=target;
    write_output;
    writeln('target setter complete.....target=',target);
    readln;
  end;
 end;


{*************************************************************************}
{************************** driver program ************************}
{*************************************************************************}
 begin
   clrscr;
   assign(pattern_file,'pattern.dat');
   rewrite(pattern_file);
   assign(target_file,'target.dat');
   rewrite(target_file);

{...initialize the counter "permutations" and read in "max permutations"}
   write('Enter maximum number of patterns desired [0..800]:',' ');  {beep;}
   readln(max_pattern_count);
   write('Enter the row dimension of your sample size[3,5,7,or 9]:',' ');
   {beep;}
   readln(samp_rows);
   samp_cols:=samp_rows;
   write('Enter minimum number of pixels which must be "on"');
   write(' to set target to 1=(true):',' ');  {beep;}
   readln(criteria);



{...setup and read in array to be analyzed}
{    readln(subj_file_name);}
   subj_file_name:='subject1.dat';
   writeln;
   assign(subj_file,subj_file_name);
   write('What is the row and column dimensions of your training image? ');
   readln(subj_rows,subj_cols);
{    subj_rows:=187;
   subj_cols:=142;}
   writeln('.....reading in the array to be analyzed.....');
   writeln;
   reset(subj_file);
   new(subj);
   data_count:=0;
   x:=WhereX;
   y:=WhereY;
```

```
    for i:=1 to subj_rows do begin
      for j:=1 to subj_cols do begin
        GotoXY(x,y);
        data_count:=data_count+1;
        write('The data count =',data_count);
        read(subj_file,subj^[i,j]);
      end;
    end;
    writeln('.....data has been read into the program.....');
    close(subj_file);
    writeln;

{...Determine first pattern and target}
    new(pattern);
    pattern_count:=0;
    sample_taker;
    pattern_setter;
    target_setter;
    write_output;
{...Set up to rotate the first pattern}
    summarizer(sum);
    samples_remaining:=max_pattern_count-pattern_count;
    if sum>0 then rotater(pattern_count,samples_remaining);
{...Set up main loop to find and generate new patterns}
    repeat
      repeat_count:=0;
      x:=WhereX;
      y:=WhereY;
      repeat
        sample_taker;
        repeat_count:=repeat_count+1;
        GotoXY(x,y);
        write('Loops in repeat check=',repeat_count);
        repeat_checker(repeat_check);
      until repeat_check>0;
      writeln;
      pattern_setter;
      target_setter;
      write_output;
      summarizer(sum);
      samples_remaining:=max_pattern_count-pattern_count;
      if sum>0 then rotater(pattern_count,samples_remaining);
      writeln;
  until pattern_count>=max_pattern_count;
{Note the end of the program}
    writeln('The program is complete -');
    writeln('Number of samples generated was:',pattern_count);
    writeln;
    write('    please press <enter> to return to the turbo pascal screen.');
    {beep;}
    readln;
    dispose(pattern);
    dispose(subj);
    close(pattern_file);
    close(target_file);
end.
```

APPENDIX III. PROGRAM PATTERNS FOR THE DEVELOPMENT OF TRAINING PATTERNS FOR A 9 X 9 SAMPLING WINDOW

```
program pattern_generator;

uses dos,crt,graph;

const
  max_patterns=800;
  max_inputs=81;
  max_size4_rows=200;
  max_size4_cols=200;
  max_size1_rows=9;
  max_size1_cols=9;

type

  boolean_array_size1=array[1..max_size1_rows,1..max_size1_cols] of boolean;
  boolean_array_size3=array[1..max_patterns,1..max_inputs] of boolean;
  boolean_array_size4=array[1..max_size4_rows,1..max_size4_cols] of boolean;
  boolean_col_vector=array[1..max_inputs] of boolean;

var
  criteria:                byte;
  data_count:                integer;
  i,j,k,l,m,n,x,y:         integer;
  max_pattern_count:           integer;
  pattern:                 ^boolean_array_size3;
  pattern_count:             integer;
  pattern_file:            text;
  repeat_check:              byte;
  repeat_count:              byte;
  samp:                    boolean_col_vector;
  samp_rows,samp_cols:         byte;
  samples_remaining:         integer;
  subj_file_name:          string;
  subj_rows,subj_cols:       byte;
  subj:                    ^boolean_array_size4;
  subj_file:               file of boolean;
  sum:                     byte;
  target:                  byte;
  target_file:             text;


{********************** procedure beep ********************************}
procedure beep;
 var
   i:    byte;
   Freq: integer;
   Time: integer;
 Begin
   Freq:=250;
   For i:=1 to 3 do begin
     Sound(Freq);
     Time:=250;
     Delay(Time);
     Nosound;
```

```
      Time:=60;
      Delay(Time)
    End;
  End;


{********************** procedure write output **********************}
 procedure write_output;
  var
    i,j,k: byte;
  begin
    k:=0;
    for i:=1 to samp_rows do begin
      for j:=1 to samp_cols do begin
        k:=k+1;
        if pattern^[pattern_count,k]=true
         then write(pattern_file,' ',1)
         else write(pattern_file,' ',0);
      end;
    end;
    writeln(pattern_file);
    writeln(target_file,target);
  end;


{************************ procedure summarizer ************************}
 procedure summarizer(var sum: byte);
  var
    i,j,k: byte;
  begin
    sum:=0;
    k:=0;
    for i:=1 to samp_rows do begin
      for j:=1 to samp_cols do begin
        k:=k+1;
        if pattern^[pattern_count,k]=true then sum:=sum+1;
      end;
    end;
  end;


{********************** procedure sample taker **********************}
 procedure sample_taker;
  var
    i,j,k,l,n: integer;
  begin
    randomize;
    i:=1+random(subj_rows-samp_rows);
    randomize;
    j:=1+random(subj_cols-samp_cols);
    n:=0;
    for k:=i to i+samp_rows-1 do begin
      for l:= j to j+samp_cols-1 do begin
        n:=n+1;
        samp[n]:= subj^[k,l];
      end;
    end;
  end;
```

```
{********************** procedure pattern_setter ************************}
procedure pattern_setter;
 var
   i,j: byte;
   k:   integer;
 begin
   pattern_count:=pattern_count+1;
   k:=0;
   for i:=1 to samp_rows do begin
    for j:=1 to samp_cols do begin
      k:=k+1;
      pattern^[pattern_count,k]:=samp[k];
      if pattern^[pattern_count,k]=true then write(chr(176),chr(176))
                                 else write(chr(178),chr(178));
    end;
    writeln;
   end;
   writeln('Exiting pattern_setter....pattern_count:=',pattern_count);
 end;


{********************** procedure repeat checker ************************}
procedure repeat_checker(var repeat_check:byte);
 var
   i:         integer;
   j,counter: byte;
   out_count: integer;
 begin
   if pattern_count=1
    then out_count:=pattern_count+1
    else out_count:=pattern_count;
   i:=0;
   repeat
     i:=i+1;
     counter:=0;
     for j:=1 to samp_rows*samp_cols do begin
       if pattern^[i,j]=samp[j] then counter:=counter+1;
     end;
    until (counter=samp_rows*samp_cols) or (i=out_count);
    if counter=samp_rows*samp_cols
     then repeat_check:=0
     else repeat_check:=1;
 end;


{***************************** target decider ***************************}
procedure target_decider;
 var
   i,j: byte;
   k: integer;
 begin
   write('Time for a target decision....Enter the Target [ 0 or 1]:');
   {beep;}
   readln(target);
 end;
```

```
{********************** target_setter procedure **********************}
 procedure target_setter;
  var
    midpoint: byte;
  begin
    sum:=0;
{    midpoint:=trunc(samp_rows*samp_cols/2+0.5);
    if (pattern^[pattern_count,midpoint]=false)
     then target:=0 else summarizer(sum);
    if (pattern^[pattern_count,midpoint]=true) and (sum<criteria) then target:=0;
    if (pattern^[pattern_count,midpoint]=true) and (sum>=criteria)
     then target_decider;    }
    target_decider;
    writeln('target setter complete.....target=',target);
    readln;
  end;


{************************ procedure rotater ****************************}
 procedure rotater(var pattern_count:integer;samples_remaining:integer);
  var
     i,j,rotation_count: byte;
     k: integer;
     matrix,rotation: boolean_array_size1;
     max_rotations: byte;

  begin

{....Set loop for max_rotations}
    if samples_remaining<3
     then max_rotations:=samples_remaining
     else max_rotations:=3;
    for rotation_count:=1 to max_rotations do begin

{....Set pattern to square matrix}
     k:=0;
     for i:= 1 to samp_rows do begin
       for j:=1 to samp_cols do begin
         k:=k+1;
         matrix[i,j]:=pattern^[pattern_count,k]
       end;
     end;

{....Take the rotation of the square matrix}
     for i:=1 to samp_rows do begin
       k:=samp_rows;
       for j:=1 to samp_cols do begin
         rotation[i,j]:=matrix[k,i];
         k:=k-1;
       end;
     end;

{....Convert rotation to a new pattern}
     writeln;
     pattern_count:=pattern_count+1;
     writeln('In rotater,  Pattern number: ',pattern_count);
     k:=0;
     for i:=1 to samp_rows do begin
```

```
        for j:=1 to samp_cols do begin
            k:=k+1;
            pattern^[pattern_count,k]:=rotation[i,j];
            if pattern^[pattern_count,k]=true then write(chr(176),chr(176))
                                    else write(chr(178),chr(178));
        end;
        writeln;
    end;

{....set the value of Target}
    target:=target;
    write_output;
    writeln('target setter complete.....target=',target);
    readln;
  end;
 end;


{*************************************************************************}
{************************** driver program ***********************}
{*************************************************************************}
 begin
   clrscr;
   assign(pattern_file,'pattern.dat');
   rewrite(pattern_file);
   assign(target_file,'target.dat');
   rewrite(target_file);

{...initialize the counter "permutations" and read in "max permutations"}
   write('Enter maximum number of patterns desired [0..800]:',' '); {beep;}
   readln(max_pattern_count);
   write('Enter the row dimension of your sample size[3,5,7,or 9]:',' ');
   {beep;}
   readln(samp_rows);
   samp_cols:=samp_rows;
   write('Enter minimum number of pixels which must be "on"');
   write(' to set target to 1=(true):',' '); {beep;}
   readln(criteria);



{...setup and read in array to be analyzed}
{    readln(subj_file_name);}
   subj_file_name:='subject1.dat';
   writeln;
   assign(subj_file,subj_file_name);
   write('What is the row and column dimensions of your training image? ');
   readln(subj_rows,subj_cols);
{    subj_rows:=187;
   subj_cols:=142;}
   writeln('.....reading in the array to be analyzed.....');
   writeln;
   reset(subj_file);
   new(subj);
   data_count:=0;
   x:=WhereX;
   y:=WhereY;
```

```
    for i:=1 to subj_rows do begin
      for j:=1 to subj_cols do begin
        GotoXY(x,y);
        data_count:=data_count+1;
        write('The data count =',data_count);
        read(subj_file,subj^[i,j]);
      end;
    end;
    writeln('.....data has been read into the program.....');
    close(subj_file);
    writeln;
{...Determine first pattern and target}
    new(pattern);
    pattern_count:=0;
    sample_taker;
    pattern_setter;
    target_setter;
    write_output;

{...Set up to rotate the first pattern}
    summarizer(sum);
    samples_remaining:=max_pattern_count-pattern_count;
    if sum>0 then rotater(pattern_count,samples_remaining);

{...Set up main loop to find and generate new patterns}
    repeat
      repeat_count:=0;
      x:=WhereX;
      y:=WhereY;
      repeat
        sample_taker;
        repeat_count:=repeat_count+1;
        GotoXY(x,y);
        write('Loops in repeat check=',repeat_count);
        repeat_checker(repeat_check);
      until repeat_check>0;
      writeln;
      pattern_setter;
      target_setter;
      write_output;
      summarizer(sum);
      samples_remaining:=max_pattern_count-pattern_count;
      if sum>0 then rotater(pattern_count,samples_remaining);
      writeln;
  until pattern_count>=max_pattern_count;
{Note the end of the program}
    writeln('The program is complete -');
    writeln('Number of samples generated was:',pattern_count);
    writeln;
    write('    please press <enter> to return to the turbo pascal screen.');
    {beep;}
    readln;
    dispose(pattern);
    dispose(subj);
    close(pattern_file);
    close(target_file);
end.
```

APPENDIX IV. THE TEMPLATE FOR THE PARAMETERS OF THE 3 X 3 SAMPLING WINDOW AS DEVELOPED BY THE PROGRAM BINARYHAM

Number of Hidden Neurons= 59 Number of elements in each pattern= 9

Patterns on each neuron of hidden layer:

```
111111111
000010000
000010011
111111100
001110000
000011100
110010000
000110101
001011010
110111011
001010100
010011000
101110111
111111110
100010001
001111111
000010111
000110110
011110111
010110010
100010110
001011001
010010110
111111110
101010010
001111111
111101101
101011101
010110000
111101111
111101111
101011111
100011010
011010100
100110010
011010001
110110101
111111100
100011000
000111001
111110011
101110111
011111100
011010010
110010000
100110100
101011101
011011111
110111011
100111000
001110100
010010101
110011111
101011000
001010000
000111010
101101111
111010111
010110000
```

Thresholds (one for each neuron)

4 5 2 2 1 1 1 1 1 2 1 1 8 6 2 8 6 0 1 1 0 1 1 7 0 1 8 6 2 7 7 1 0 7 1 1 8 0 8 1 8 0 8 1 8 0 8 0 8 1 8 1 8 1 8 1 8 0 8

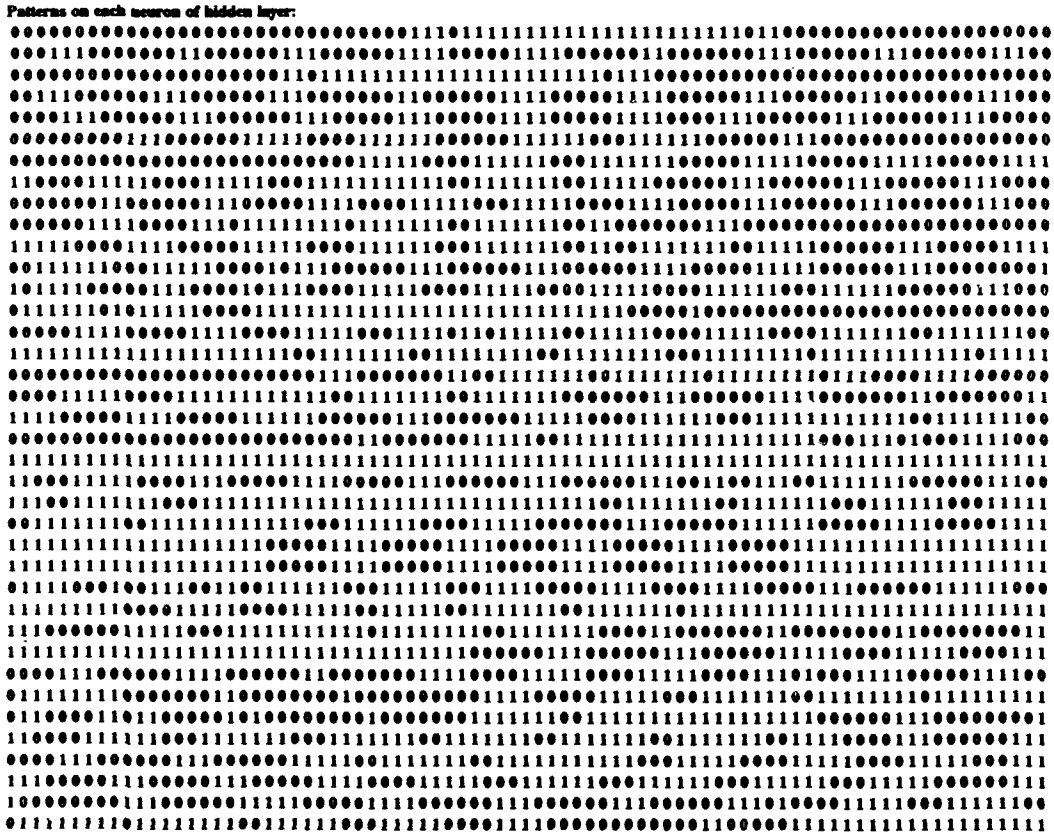
Threshold on the output neuron:

23

APPENDIX V. THE TEMPLATE FOR THE PARAMETERS OF THE 9 X 9 SAMPLING WINDOW AS DEVELOPED BY THE PROGRAM BINARYHAM

Number of Hidden Neurons= 38 Number of elements in each pattern= 81

Patterns on each neuron of hidden layer:



Thresholds on each hidden neuron:

19 19 18 18 23 23 18 18 18 14 15 14 17 17 15 59 14 17 19 12 21 20 80 19 80 80 20 80 20 80 12 80 20 80 19 80 14 80

Threshold on output neuron:

10

APPENDIX VI. NOTATION.

a: Activation value of a neuron
b: bias. This value determines the lateral placement of the sigmoid transfer function with respect to the axis of the dependent variable(s).
t: threshold value for a transfer function
v, w: connection constants, i.e., weights, offsets, etc.
f: Symbol for the transfer function of a neuron
Σ: Summation

APPENDIX VII. REFERENCES

Bailey, D., and Thompson, D. (1990) "How to Develop Neural-Network Applications." AI Expert, 38-47.

Castelaz, P., Angus, J., and Mahoney, J. (1987) "Application of Neural Networks to Expert Systems and Command and Control Systems." Proc., IEEE Western Conf. on Expert Systems, 118-125.

Derthick, M. (1987) "A Connectionist Architecture for Representing and Reasoning about Structured Knowledge." Proc., Ninth Annual Conf. Cognitive Science Soc.

Fahlman, S., and Hinton, G. (1986) "Connectionist Architectures for Artificial Intelligence." Computer,19,100-108.

Gustaferro, A., Scott, N. (1990). "Reading Structural Concrete Cracks." Concrete Construction, (Dec), 994-1003.

Moselhi, O., Hegazy, T., and Fazio, P. (1991). "Neural Networks as Tools in Construction." Journal of Construction Engineering and Management, 117(4), 606-625.

Flood, I. (1989). "A Neural Network Approach to the Sequencing of Construction Tasks." Proc., 6th Int. Symp. on Automation and Robotics in Construction, ISARC, 204-211.

Flood, I. (1990). "Solving Construction Operational

Problems Using Artificial Neural Networks and Simulated Evolution." Proc., Int. Symp. on Building Economics and Construction Management, CIB, Sydney, Australia, 197-208.

Garret, J., et al. (1991) "Engineering Applications of Neural Networks." Journal of Intelligent Manufacturing. (Accepted for Publication February 2, but yet unpublished).

Gorman, R., and Sejnowski, T. (1988) "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets." Neural Networks, v.1, 75-89.

Lippman, R., and Beckman, P. (1989) "Adaptive Neural-Net Processing for Signal Detection in Non-Gaussian Noise." Advances in Neural Information Processing Systems, 1, D. Touretzky, ed., San Mateo, California: Morgan Kauffman.

Lippman, R. (1987). "Review of Neural Networks for Speech Recognition." Neural Computing, v.1(1), 1-38.

Minsky, M., and Papert, S., (1969) Perceptions. MIT Press, Cambridge, Massachusetts.

Mohan, S. (1990) "Expert Systems Applications in Construction Management and Engineering." J. Constru. Engrg. Mgmt., ASCE, 116(1), 87-99.

Qian, N., Sejnowski, T. (1988) "Predicting the Secondary Structure of Globular Proteins using Neural Network Models." J. Molecular Biology, v.202, 865-884.

Roth, M. (1990) "Survey of Neural Network Technology for Automatic Target Recognition." IEEE Trans. Neural Networks, v.1, no. 1 (Mar), 28-43.

Rumelhart, D., and McClleland, J., et al (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volumes 1 and 2. MIT Press, Cambridge, Massachusetts.

Sejnowski, T., Koch, C., and Churchland, P. (1988). "Computational Neuroscience". Science, v.241, no. 4871, 1299-1305.

Sejnowski, T., Rosenberg, C. (1987) "Parallel Networks that Learn to Pronounce English Text." Complex Systems, v.1, 145-168.

"Specialized Concrete Evaluation and Testing." (1984) Concrete Construction (Dec), 1097-1100

Tamura, S., and Waibel, A. (1988) "Noise Reduction using Connectionist Models." Proc. IEEE Int'l. Conf. Acoustics, Speech, and Signal Processing, April, 553-556.

Tesauro, G., and Sejnowski, T. (1988) "A Neural Network that Learns to Play Backgammon." Neural Information Processing Systems, D. Anderson, 794-803, American Institute of Physics, New York.

Udpa, L., and Udpa, S. (1991) "Neural Networks for the Classification of Nondestructive Evaluation Signals." IEE Proceedings-F, v. 138, No. 1 (Feb), 41-45.

Weideman, W., Manry, M., and Yau, H. (1989) "A Comparison of a Nearest Neighbor Classifier and a Neural Network for Numeric Handprint Character Recognition." Proc. Int. Joint Conf. Neural Networks, 117-120.